Technical Paper
Open Access
CrossMark

Artificial neural networks-based simulation of obstacle detection with a mobile robot in a virtual environment

Abstract

Mobile robot navigation is primarily a task that occurs in a real environment. However, simulating obstacles and robot movements in a virtual environment can provide significant advantages and yield good results, as demonstrated in this paper. By employing artificial neural networks (ANNs), it is possible to develop a trained system in a virtual environment that can detect obstacles using data collected from various sensors. In this study, infrared (IR) sensors and a camera were utilized to gather information from the virtual environment. The MatLab Simulink software package was used as a tool to train the artificial neural networks. Detection and avoidance of obstacles were simulated in the RobotinoSIM virtual environment.

Keywords: obstacle detection, mobile robot, artificial neural networks, virtual simulation

Volume 9 Issue 2 - 2023

Boris Crnokić,[1] Ivan Peko,[2] Miroslav Grubišić[3]
[1,3]University of Mostar, Faculty of Mechanical Engineering, Computing and Electrical Engineering, Bosnia and Herzegovina
[2]University of Split, Faculty of Science, Croatia

Correspondence: Boris Crnokić, University of Mostar, Faculty of Mechanical Engineering, Computing and Electrical Engineering, Matice hrvatske b.b., 8800 Mostar, Bosnia and Herzegovina, Email boris.crnoki@fsre.sum.ba

Received: April 29, 2023 | **Published:** May 16, 2023

Introduction

Obstacle detection is one of the fundamental steps in the process of autonomous robot movement and navigation through an unknown environment. Various types of sensors are used to collect information from the environment. Based on the data obtained from the sensors, different processing, decision-making, and information implementation systems are built, enabling the robot to detect obstacles and safely navigate the environment. Artificial neural networks (ANNs) are a common tool for this task, and different structures and types of ANNs are used to implement various mobile robot navigation systems.[1–3] In these systems, infra-red sensors (IR) and various robotic vision sensors, either independently or in fusion, are frequently used.[4–7] Paper[8] presented a neural network-based obstacle avoidance robot using low-cost IR sensor arrays, where MatLab was used to train the neural networks. In paper,[5] modeling and simulation of obstacle detection and avoidance with a four-wheel mobile robot using a Deep Neural Network were carried out using MatLab Simulink's robotic toolbox and robotic operating system toolbox. MatLab was also used for the development and implementation of neural control systems in mobile robots for obstacle avoidance in real-time using ultrasonic sensors,[9] and to train a mobile robot to avoid obstacles using range sensor readings that detect obstacles in the map.[10] Obstacle detection systems can be tested in a real environment or through simulation frameworks, which can reduce experimental costs, time, and the risks of negative aftereffects of accidents.[11–13] Moreover, more and more research focuses on the application of virtual environments with obstacles that enable reliable robot simulation. [14,15] While the application of artificial neural networks in such systems has both advantages and disadvantages, they still provide satisfactory results for obstacle detection.[16]

This paper demonstrates the application of artificial neural networks for obstacle detection in a virtually simulated process of robot movement and obstacle avoidance in a virtual environment. The proposed system uses the readings from three IR sensors and a camera. The MatLab software package was used to train the artificial neural network and control the robot. The RobotinoSIM virtual simulation environment was used to perform all movement and obstacle avoidance tasks by the Robotino mobile robot.

The proposed system

Structure of the proposed system

As demonstrated in paper,[17] the proposed system structure is divided into three segments: Perception, Reasoning, and Reacting. In the "Perception" section, data is collected and pre-processed from three infrared sensors and a camera. The system processes the data received from the sensors, compares it with the required reference input values in the "Reasoning" section, and makes a judgment for the robot's output movements. Finally, the "Reacting" section provides the final calculations to take the necessary actions, such as robot movement or obstacle avoidance. Figure 1 depicts the system structure.

Figure 1 The proposed system structure: Perception, Reasoning, and Reacting.

The programming was done in the MatLab software package, while the robot simulation was performed in the RobotinoSIM virtual environment.[18] A virtual model of the Robotino mobile robot was utilized to perform the tasks of detecting and avoiding obstacles. The Robotino mobile robot is capable of omnidirectional movement through the environment and has an external sensor system comprising a stereo camera, 9 IR sensors, and bumpers in the basic configuration. For the experiment presented in this paper, a Logitech C250 camera was used to gather data (images and videos) from the environment, and three SHARP GP2D120 IR sensors (IR1, IR2, and IR9) were employed to measure the distance from obstacles in front of the robot.[19]

Obstacle detection is carried out in two parts: for the camera readings and for the IR sensor readings. The camera is employed for landmark detection and localization, such as detecting edges, overhangs, and robot movement space. IR sensors are employed to measure the distance from obstacles and detect them in real-time. The Canny method combined with LPQ description was utilized for image preprocessing, such as edge detection and feature extraction.

An ANN performs the task of "teaching the system" to recognize obstacles in the environment by using information obtained from the camera. Only certain obstacles were taken for training the network, which greatly reduced the computational complexity of the control system. After the training, the system is able to decide whether the robot should continue moving forward (if one of the learned obstacles is not detected), or to avoid the obstacle (if one of the obstacles is detected). Additional information about detected obstacles (from 3 IR sensors). Based on this information, the robot has a complete set of necessary information in order to make a final decision on which actions to take.

Acquisition of information from the environment

The Robotino View program[20] was used to collect initial images for network training and to set up video and image resolution values. Robotino View has a simple interface for connecting to Robotino and using tools to collect and save images and videos, as shown in Figure 2a, Figure 2b shows the position of Robotino when it detected the obstacle. The Control Panel block can be used to bring the robot to any position in the environment. The Camera block activates the camera module, and the Image Writer block is used to save images. In this module, two image resolutions, 320 x 240 and 640 x 480, can be selected, as shown in Figure 2c. The resolution of the collected images for neural network training in this experiment is 320 x 240.

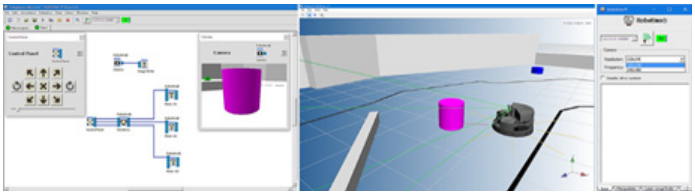

Figure 2 a) Image collecting in Robotino View program, b) Encountering an obstacle in the RobotinoSIM environment, c) Adjusting the image resolution.

After connecting the robot to the MatLab interface using the IP address '127.0.0.1:8080', the real-time image collection process is enabled by using the following commands:

```
CameraId = Camera_construct;
Camera_setComId (CameraId, ComId);
if ~(Camera_setStreaming(CameraId, 1) == 1)
  disp ('Camera_set Streaming failed.');
  end;
if (Camera_grab(CameraId) == 1)
 img = Camera_getImage( CameraId );
  end
```

Obtaining information from the three IR sensors is done by using the following commands:

```
DistanceSensor0Id = DistanceSensor_construct (0);
DistanceSensor1Id = DistanceSensor_construct (1);
DistanceSensor8Id = DistanceSensor_construct (8);
value0 = DistanceSensor_voltage(DistanceSensor0Id)
 value1 = DistanceSensor_voltage(DistanceSensor1Id)
 value8 = DistanceSensor_voltage(DistanceSensor8Id)
 if((0.7 <= value0) | (0.7 <=value1) | (0.7 <= value8)) & (class==1)
```

The three infrared sensors are labeled as DistanceSensor0, DistanceSensor1, and DistanceSensor8, respectively, and are used to obtain information. A voltage of 0.7 V, which corresponds to a distance of approximately 17 cm, was set as the reference distance for obstacle detection. If the voltage values obtained from the sensors are lower than this value, it means that an obstacle has been detected, and the task of avoiding the obstacle will be activated. Conversely, if the voltage values are higher, the system will assume that no obstacle is present.

Edge detection and feature extraction from images

In the process of detecting edges in images with obstacles, the Canny edge detection method was used. Images were obtained from the camera, and before applying the Canny method, they were converted from an RGB image to a grayscale image. All these operations were performed using the Image Processing Toolbox within MatLab. The part of the algorithm responsible for these image pre-processing operations is as follows:

```
function [yind]=img_class(img)
load lpqtrain
img=rgb2gray(img);
canny_filtrd_img=edge(img,'canny');
feature_v=(lpq(canny_filtrd_img))';
y = net(feature_v);
yind = vec2ind(y);
end
```

In addition to the Canny filter, we used the LPQ (Local Phase Quantization)[21] descriptor for image pre-processing and feature extraction. The downloaded images were in matrix form, and after processing, they needed to be transformed into a vector column for further use in the neural network training process.

Avoidance of obstacles detected by IR sensor

The part of the algorithm responsible for detecting and avoiding of obstacles is as follows:

```
while((Bumper_value(BumperId) ~= 1)) && stoppressed==0
   OmniDrive_setVelocity(OmniDriveId, 100, 0 ,0)
   tElapsed = toc(tStart);
    if(tElapsed >= 60 )
      break;
   end;
   value0 = DistanceSensor_voltage(DistanceSensor0Id)
   value1 = DistanceSensor_voltage(DistanceSensor1Id)
   value8 = DistanceSensor_voltage(DistanceSensor8Id)
    if((0.7 <= value0)|(0.7 <=value1)|(0.7 <= value8))&(class==1)
      % Obstacle approach
      break;
   else
      % There is no obstacle in front of the robot
      OmniDrive_setVelocity(OmniDriveId, 100, 0 ,0);
   end;
end
 OmniDrive_setVelocity(OmniDriveId, 0, 0 ,0);
 delay(1)
 OmniDrive_setVelocity(OmniDriveId, 0, 100 ,0);
 delay(3)
 OmniDrive_setVelocity(OmniDriveId, 100, 0 ,0);
 delay(4)
 OmniDrive_setVelocity(OmniDriveId, 0, -100 ,0);
 delay(3)
 OmniDrive_setVelocity(OmniDriveId, 100, 0 ,0);
```

The obstacle avoidance vector is determined based on whether any of the three IR sensors has detected an obstacle. Accordingly, two velocities are defined:

- v_x in the direction of the robot's movement along the ***x axis***, and
- v_y in the direction of the robot's movement along the ***y axis***.

In the previous code, these velocities, as well as the robot's angular velocity ω, are represented in the following format:

```
OmniDrive_setVelocity(OmniDriveId, vx, vy ,ω)
```

The velocities are expressed in *mm/s*. Depending on the direction in which the obstacle is detected, their values can be 0, 100, or -100 mm/s. The robot can move in the following ways depending on which sensor detects the obstacle:

- ***forward***, for $v_x = 100$, $v_y = 0$ (if no obstacle is detected in front of the robot),
- ***right***, for $v_x = 0$, $v_y = 100$ (if an obstacle is detected to the left of the robot), and
- ***left***, for $v_x = 0$, $v_y = -100$ (if an obstacle is detected to the right of the robot).

Due to the simplification of the robot's movements when avoiding obstacles, the angular velocity of the robot was not applied. To rotate the robot when avoiding obstacles, all 9 IR sensors must be active, allowing detection of all obstacles within a 360° radius around the robot. Since only 3 IR sensors are active in the proposed system, it is impossible to guarantee that the robot will not collide with an obstacle that was not detected by one of the 3 active sensors during a possible rotation.

Artificial neural network

The design of the ANN is realized through the following 7 steps:

- Collecting data,
- Creating the neural network,
- Configuring the neural network,
- Initializing the weights and biases,
- Training the neural network,
- Validating the neural network (analysis after learning), and
- Using the neural network.

Before starting the process of designing a neural network, it is necessary to prepare data samples correctly and with high quality for learning the network. It is difficult to incorporate prior knowledge into the network, so the network will only be as precise and accurate as the information used to train it. It is essential that the information used for training covers the complete range of inputs for which the network is used. After collecting the data, two steps must be taken before using it for training:

- Pre-processing of the data is required (Canny + LPQ), and
- The data must be divided into different subsets.

For training the neural network in this experiment, 36 pictures of obstacles were taken:

```
x = csvread('training_featuressslpq.csv');
```

For testing, 9 more pictures of the same obstacles were taken:

```
tst_inputs=csvread('testing_featuressslpq.csv');
```

The obstacles are classified into three different categories, with 12 pictures in each category. The entire set of training images is then divided into three subsets as follows:

- 70/100 (70%) for *training*:

  ```
  net.divideParam.trainRatio = 70/100;
  ```

- 15/100 (15%) for *validation* (used to validate that the network is generalized and to stop training before overtraining):

  ```
  net.divideParam.valRatio = 15/100;
  ```

- 15/100 (15%) for *testing* (used as a completely independent test of network generalization):

  ```
  net.divideParam.testRatio = 15/100;
  ```

The artificial neural network used in this experiment is a multilayer perceptron feedforward network for pattern recognition, with one input, one output, and one hidden layer:

```
net = patternnet(hiddenLayerSize);
```

This type of ANN can be trained to classify patterns according to a given target data set, and it is initiated in MATLAB with the

'patternnet' function. The target value data of the pattern recognition network consists of vectors where all values are 0 except for one element which has a value of 1, where "i" is the class it represents. The number of input and output nodes was selected in relation to the requirements and functionality of the proposed system. The selection of hidden layer nodes is more complex and comes down to the "trial-and-error" method (i.e. testing the number of hidden nodes and comparing them with the performance of the neural network). The following structure was defined for the artificial neural network in this experiment:

- The *input layer* consists of 256 input units (nodes) representing the image obtained after preprocessing with the LPQ algorithm. These are inputs from the LPQ descriptor formed from a histogram of integer values of all image positions, and is used as a 256-dimensional vector of image features. This represents 256 hypercubes in eight-dimensional space, resulting in an 8-bit code. Since 36 images were taken for training, the network input is essentially a 256 x 36 matrix.
- The *hidden layer* has 10 nodes.
- The *output layer* has 3 output nodes, where the nodes represent outputs that specify one of three classes of input images.

When training the network, the target values are defined (the correct output vector ***t*** for each of the input vectors, where the target vector values are defined for each of the classes):

t = [1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3];

The target vector is a binary value with the index of correct classes defined as 1, or with the index of all other classes (i.e. incorrect classes) defined as 0. The target values are represented by a 3 x 36 matrix. Target values in the case of network testing are represented by a 3 x 9 matrix, with the same characteristics as the previous one:

t=[1 1 1 2 2 2 3 3 3];

Before training the ANN, initialization is necessary. For this neural network, the initialization operation is performed automatically by MATLAB each time the training process starts. When training the network for pattern recognition, the following three functions are most commonly used: 'trainlm', 'trainbr', and 'trainscg'. In most cases, the 'trainlm' function proves to be the fastest, while 'trainbr' is a slower function but a better choice for very demanding problems. The 'trainscg' function takes up the least memory resources and is suitable for cases where memory is limited. In this experiment, the 'trainscg' learning function representing the SCG algorithm was used to train the ANN:

TrainFcn = 'trainscg';

[net, tr] = train(net, x, targets);

During the training process of the network, the progress is constantly updated in the "nntraintool" window, as shown in Figure 3.

"Random" distribution was selected for data distribution using the 'dividerand' command. As mentioned earlier, the SCG algorithm was used for training. To analyze the performance of the neural network, the "Cross-Entropy" function was used via the "crossentropy" command. This function calculates the network performance for a given input and output values with the ability to select performance weights and other parameters. In the ANN training monitoring window, it is also possible to monitor the number of iterations ("Iterations"), i.e., the steps in the algorithm in which the weights are adjusted, as well as the entire epoch ("Epoch"), which shows one representation of the entire learning set. There is also the possibility of monitoring the learning time of the network with the "Time" function.

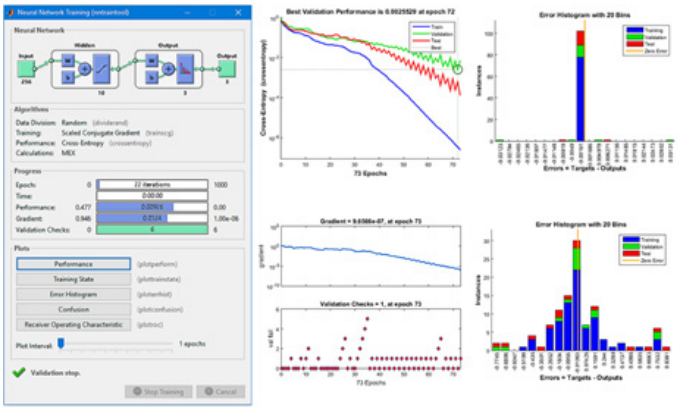

Figure 3 Monitoring the training process of the artificial neural network.

After repeating the network training process, it is possible to obtain different results for each case due to different initial values of the weights and different data distributions in the subsets for learning, validation, and testing. As a result, different neural networks trained for the same problem can produce completely different output values for the same input values. To avoid this, i.e., to ensure a network that will be as accurate as possible, it is necessary to train the network several times. For this reason, this network was also trained through several repetitions.

GUI for MatLab

To control the movement of the robot and the functions of the proposed system from the MatLab environment, a separate GUI was created. The GUI allows the control algorithm to start, as well as individual control of the robot's movements in different directions. It also includes all the previously mentioned functions of gathering information from sensors, detecting obstacles, and undertaking tasks of moving and avoiding obstacles. Figure 4 shows the GUI.

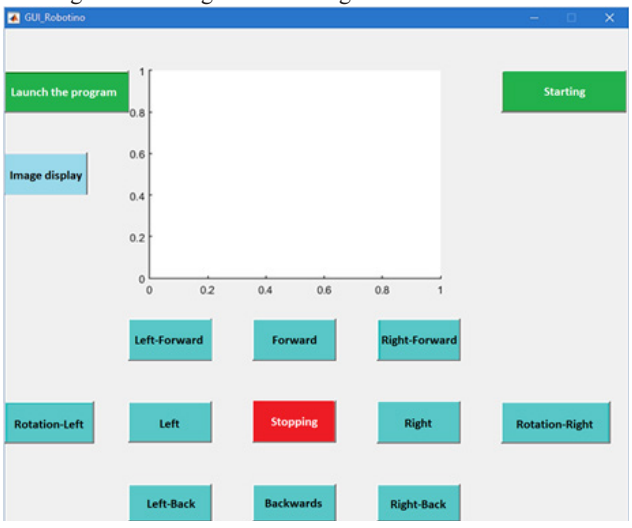

Figure 4 GUI for MatLab.

Controlling the basic movements of the robot (forward, backward, left, right, etc.) is possible after executing the "Start" command. This function was created to enable the robot to be positioned in front of an obstacle before testing the control algorithm. The algorithm is activated by executing the "Launch program" command. This

command triggers the robot's autonomous movement and obstacle avoidance capabilities. Another function is "Image display," which allows for real-time display of images captured by the robot's sensors.

Results analysis

Various types of obstacles, including rollers of different colors, walls, boxes, and lines on the floor, (Figure 5) were used in the RobotinoSIM virtual environment. For each obstacle, 20 detection attempts were made, and successful and unsuccessful attempts were recorded. The robot approached the obstacles from different sides, angles, and distances.

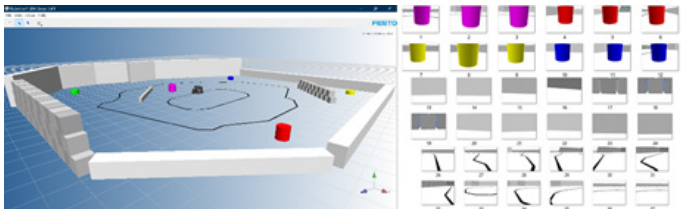

Figure 5 Obstacles in RobotinoSIM virtual environment.

In the MatLab command line it is possible to see which of the sensors detected an obstacle. If one of the IR sensors detects an obstacle, voltage values will appear in the command line (value0, value1, value8), as shown in Figure 6a. If the camera detects an obstacle, the class from which the neural network classified a particular obstacle will be displayed (class = 1, class = 2, class = 3). (Figure 6b)

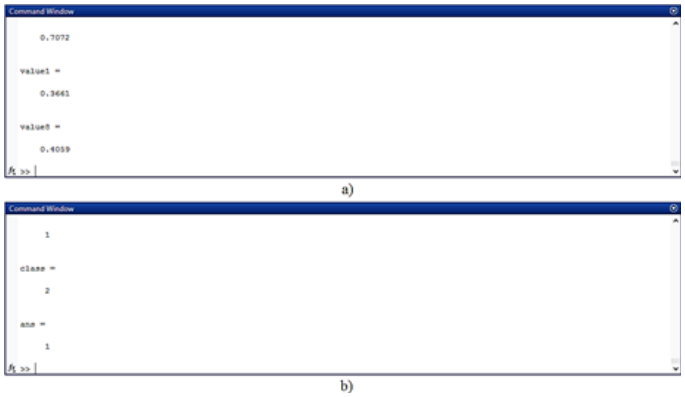

Figure 6 The MatLab command line displays information from various sensors.

A total of 180 obstacle detection attempts were made. In 154 attempts, some of the obstacles were successfully detected, while 26 attempts ended unsuccessfully, i.e., the obstacles were not detected. Such a ratio of detected and undetected obstacles resulted in an 85.56% accuracy of obstacle detection. Since the system is based on the fusion of a camera and infrared sensors, the ratio of the share of individual sensors in detecting obstacles is also interesting. The three IR sensors detected the obstacle 98 times (63.64%), while the camera detected obstacles 56 times (36.36%). It is not possible to manipulate obstacles in the RobotinoSIM environment using a specific command. However, it was possible to overturn some of the obstacles with the robot's movements. Overturned obstacles had a different appearance, depending on the position they remained after overturning. These overturned obstacles became a good test for the system since the neural network was not trained to detect such "new" obstacles.

Conclusion

In line with previous experiences, this experiment has also demonstrated that obstacle detection and avoidance is a highly complex task. However, the results of obstacle detection can ultimately be satisfactory. When detecting all obstacles, the system has proven to be just as accurate as when detecting obstacles presented to the ANN during the training process. The obtained results are a good indication that the neural network was not over trained, i.e., the network did not become overly specialized in detecting only samples from the training, validation, and testing sets. Artificial neural networks have proven to be a powerful tool for use in such environments where input information is not always unambiguous and of the same character. The structure and type of the neural network, as well as all the conditions and parameters of the training in this experiment, have proven to be adequate for the tasks of detecting obstacles. The classification capabilities of ANN have proven to be excellent, even on examples of obstacles not presented during the training process.

However, the accuracy of obstacle avoidance is significantly limited by the use of only three infrared sensors, as these sensors cover an angle of only 80º in front of the robot. Additionally, there is the problem of camera immobility, as the camera can only receive information about obstacles that are directly in front of the robot, making it impossible to detect obstacles on the sides and behind the robot. Due to all the aforementioned limitations, Robotino can avoid obstacles only by moving forward, left, or right.

The disadvantage of using artificial neural networks is the demanding process of collecting and preparing input datasets that will provide the most accurate information required for optimal implementation of the network training process. Moreover, for greater classification accuracy, it is necessary to present larger datasets, which can ultimately result in longer learning and data processing times, as well as higher computer requirements.

Sensor fusion has proven to be a much better solution than the independent use of IR sensors or cameras. Further research may include upgrading the system with other types of sensors, such as ultrasound, laser scanners, 3D or thermal cameras, etc. By including one of the mentioned sensors, it will be possible to replace the shortcomings of the camera and IR sensor and, with their fusion, raise the level of navigation accuracy and improve system performance.

Acknowledgements

None.

Conflicts of interests

Author declares that there is no conflict of interest.

References

1. Andreev V, Tarasova V. *The mobile robot control for obstacle avoidance with an artificial neural network application*. Ann DAAAM Proc Int DAAAM Symp. 2019;30(1):724–732.
2. Farag KKA, Shehata HH, El-Batsh HM. Mobile robot obstacle avoidance based on neural network with a standardization technique. *J Robot*. 2021;2021.
3. Lee HY, Ho HW, Zhou Y. Deep Learning-based monocular obstacle avoidance for unmanned aerial vehicle navigation in tree plantations: Faster region-based convolutional neural network approach. *J Intell Robot Syst Theory Appl*. 2021;101(1).
4. Rezaei N, Darabi S. Mobile robot monocular vision-based obstacle avoidance algorithm using a deep neural network. Evol Intell; 2023.
5. Eneh Princewill C, Eneh Innocent I, Egoigwe Sochima V, et al. Deep artificial neural network based obstacle detection and avoidance for a non-holonomic mobile robot. 2019;16(3).

6. Khan MO, Parker GB. Vision based indoor obstacle avoidance using a deep convolutional neural network. *Science and Technology Publications*. 2019;403–411.
7. Farias G, Fabregas E, Peralta E, et al. A neural network approach for building an obstacle detection model by fusion of proximity sensors data. *Sensors (Basel)*. 2018;18(3):683.
8. Nagarani R, Nithyavathy N, Parameshwaran R. Lowcost mobile robot using neural networks in obstacle detection; 2013.
9. Medina-Santiago A, Camas-Anzueto JL, Vazquez-Feijoo JA, et al. Neural control system in obstacle avoidance in mobile robots using ultrasonic sensors. *Journal of Applied Research and Technology*. 2014;12(1):104–110 .
10. The MathWorks Inc. Avoid obstacles using reinforcement learning for mobile robots-MATLAB &Simulink.
11. Faizullin RV. Simulator of the navigation equipped with LIDAR of the mobile robot based on the neural network. *Materials Science and Engineering*; 2020.
12. Antúnez E, Palomino AJ, Marfil R, et al. Perceptual organization and artificial attention for visual landmarks detection. *Cogn Process*. 2013;14(1):13–18.
13. Crnokić B, Grubišić M. *Fusion of infrared sensors and camera for mobile robot navigation system-simulation scenario*. In Proceedings of 13th International Scientific Conference Novi Sad; Serbia. 2018;28-29:71–75.
14. Li Y, Dai S, Shi Y, et al. Navigation simulation of a mecanum wheel mobile robot based on an improved A* Algorithm in unity3D. *Sensors (Switzerland)*. 2019;19(13):2976.
15. Ngwenya T, Ayomoh M, Yadavalli S. Virtual obstacles for sensors incapacitation in robot navigation: A systematic review of 2D path planning. *Sensors (Basel)*. 2022;22(18):6943–6943.
16. Verbitsky NS, Chepin EV, Gridnev AA. Experimental studies of a convolutional neural network for application in the navigation system of a mobile robot. *Procedia Comput Sci*. 2018;145:611–616.
17. Crnokić B. Infrared and vision sensors a mobile robot navigation system; 2020.
18. Festo.com. Robotino SIM -Robotino SIM - Other training software - Digital Learning - Learning Systems - Festo Didactic; 2023.
19. Festo Didactic. Robotino® Mobile robot platform for research and training. Denkendorf: 56940; 2013.
20. Festo.com. Robotino® View-Programming-Robotino®-Services- Festo Didactic; 2023.
21. Pedone M, Heikkilä J. *Local phase quantization descriptors for blur robust and illumination invariant recognition of color textures*. Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012); 2012.

Citation: Crnokić B, Peko I, Grubišić M. Artificial neural networks-based simulation of obstacle detection with a mobile robot in a virtual environment. *Int Rob Auto J*. 2023;9(2):62–67. DOI: 10.15406/iratj.2023.09.00265